Petite mise à jours dans Compte.php

Créer fonction retirer

```php
/**
 * Retire un  montant du solde du compte
 *
 * @param float $montant
 * @return void
 */
public function retirer(float $montant)
{
    // On vérifie si le découvert permet le retrait
    if($montant > 0 && $this->solde - $montant >= $montant)
    {
        $this->solde -= $montant;
    }else{
        echo 'Montant invalide ou solde insuffisant';
    }
}
```

Toutes les variables $solde excepté celle protected
on été changé par $montant pour faciliter la compréhension

```php
abstract class Compte
{
    /**
     * Titulaire du compte
     * @var string
     */
    private $titulaire;

    /**
     * Solde du compte
     * @var float
     */
    protected $solde;

     /**
     *
     *Constructeur de compte bancaire
     * @param string $nom du titulaire
     * @param float $montant Montant du solde à l'ouverture
     */
    public function __construct(string $titulaire, float $montant = 500)
    {
        // On affecte le titulaire à la propriété titulaire
        $this->titulaire = $titulaire;

        // On affecte le solde à la propriété solde
        $this->solde = $montant;
    }
```

L’héritage

L'un des concepts du développement Orienté Objet est la possibilité de faire dépendre certains objets d'autres objets "parents".

Dans les articles précédents, nous avons créé un objet "**Compte**" qui permet de **créer un compte bancaire**.

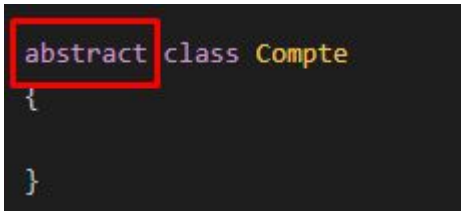
Il existe cependant plusieurs types de comptes bancaires, certains offrent une **possibilité de découvert**, d'autres permettent de **toucher des intérêts**.

Nous allons donc modifier notre objet "**Compte**" pour pouvoir gérer des "**sous-objets**" qui en hériteront.

Pour éviter d'instancier notre classe "**Compte**", nous allons la passer en classe abstraite par le mot clé "**abstract**".

Elle ne pourra donc plus être instanciée directement.

Compte.php



De plus, toutes les propriétés qui pourront être manipulées dans le "**sous-objet**" devront être en visibilité "**public**" ou "**protected**". En effet, les propriétés en visibilité "**private**" ne sont pas disponibles par héritage.

Compte.php

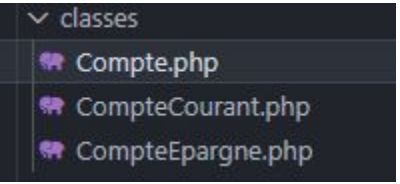
```php
/**
 * Titulaire du compte
 * @var string
 */
protected $titulaire;

/**
 * Solde du compte
 * @var float
 */
protected $solde;
```

L'héritage

Nous allons créer deux classes qui hériteront de la classe "**Compte**". Elles s'appelleront "**CompteCourant**" et "**CompteEpargne**" dans les fichiers du même nom.

Création des fichiers dans dossier classes



CompteCourant.php

```php
class CompteCourant extends Compte
{

}
```

CompteEpargne.php

```php
class CompteEpargne extends Compte
{

}
```

Nous devrons, pour le moment, appeler chacun des fichiers avant de pouvoir les utiliser. Nous verrons dans l'article suivant une méthode pour charger automatiquement tous les fichiers nécessaires.

index.php

```php
require_once 'classes/Compte.php';
require_once 'classes/CompteCourant.php';
require_once 'classes/CompteEpargne.php';
```

Le compte courant

Focalisons nous sur le **compte courant**. C'est un compte bancaire dans lequel nous allons **autoriser un découvert**.

Son **constructeur** nécessitera **au moins les informations du compte bancaire** (titulaire et solde) et **en option un découvert**, mis à 500 CAD par défaut.

Le constructeur sera donc le suivant

```php
/**
 * Découvert autorisé
 * @var int
 */
private $decouvert;

/**
 * Constructeur du compte courant
 * @param string $titulaire Titulaire du compte
 * @param float $solde Solde du compte
 * @param int $decouvert Découvert autorisé
 * @return void
 */
public function __construct(string $titulaire, float $solde, int $decouvert
= 500)
{
    // On appelle le constructeur du parent
    parent::__construct($titulaire, $solde);

    // On définit les propriétés "locales"
    $this->decouvert = $decouvert;
}
```

Nous pourrons donc **instancier un compte courant** de cette façon

index.php

```php
// On instancie sans préciser le découvert
$compte1 = new CompteCourant('Benoit', 2000);

// On instancie en précisant le découvert
$compte2 = new CompteCourant('Benoit', 2000, 200);
```

Etant donné que notre compte devra **bénéficier d'un découvert**, nous pourrons avoir **un solde négatif** jusqu'au montant du découvert.

Nous devrons donc modifier la méthode "**retirer**" pour le "**CompteCourant**" spécifiquement.

Nous écrirons donc

```php
public function retirer($montant){
    // On vérifie si le découvert permet le retrait
    if($this->solde - $montant > -$this->decouvert){
        $this->solde -= $montant;
    }else{
        echo 'Solde insuffisant';
    }
}
```

Enfin, la propriété "**decouvert**" étant privée, nous allons écrire ses accesseurs (Getter et Setter)

```php
public function getDecouvert()
{
    return $this->decouvert;
}

public function setDecouvert(int $decouvert) : self
{
    $this->decouvert = $decouvert;

    return $this;
}
```

Le compte épargne

Pour le compte épargne, nous allons procéder de façon similaire afin de mettre en place le **versement des intérêts**.

Nous aurons donc une propriété "**taux_interets**" qui correspondra au taux d'intérêts du compte et un **constructeur** en conséquence. Nous allons choisir de **rendre obligatoire la déclaration de ce taux** dans le constructeur.

```php
/**
 * Taux d'intérêts
 * @var int
 */
private $taux_interets;

/**
 * Constructeur du compte courant
 * @param string $titulaire Titulaire du compte
 * @param float $solde Solde du compte
 * @param int $taux Taux d'intérêts du compte
 * @return void
 */
public function __construct(string $titulaire, float $solde, int $taux)
{
    // On appelle le constructeur du parent
    parent::__construct($titulaire, $solde);

    // On définit les propriétés "locales"
    $this->taux_interets = $taux;
}
```

CompteEparge.php

Enfin, pour verser les intérêts, nous écrivons une méthode comme ceci

```php
public function verserInterets(){
    if($this->solde > 0){
        $this->solde = $this->solde + ($this->solde * $this->taux_interets /
100);
    }else{
        echo 'Solde insuffisant';
    }
}
```

Sans oublier les accesseurs du taux d'intérêts

```php
public function getTauxInterets()
{
    return $this->taux_interets;
}

public function setTauxInterets(int $taux) : self
{
    $this->taux_interets = $taux;

    return $this;
}
```

Index.php

ritages > index.php > ...

C:\wamp64\www\LaPooPHP\3 Héritages • Contains emphasized items

```php
require_once 'classes/Compte.php';
require_once 'classes/CompteCourant.php';
require_once 'classes/CompteEpargne.php';

// Instanciation de CompteCourant.
$compte1 = new CompteCourant('Sam', 500, 200);
// call methode retirer de la class comptecourant  solde 500 - 15 = 485.
// A tester retiré 600 - 500 = solde insufisant.
$compte1->retirer(15);
$compte1->getDecouvert();
var_dump($compte1);


$compte2 = new CompteEpargne('bob', 200, 1);
// methode retirer de  la class Compte
$compte2->retirer(15);
echo 'taux intéret ' . $compte2->getTauxInterets();
var_dump($compte2);

$compte2->verserInterets();

var_dump($compte2);
```

```
C:\wamp64\www\LaPooPHP\3 Héritages\index.php:10:
object(CompteCourant)[1]
  private 'decouvert' => int 200
  private 'titulaire' (Compte) => string 'Sam'
  protected 'solde' => float 485
```

taux intéret 1

```
C:\wamp64\www\LaPooPHP\3 Héritages\index.php:16:
object(CompteEpargne)[2]
  private 'tauxInterets' => int 1
  private 'titulaire' (Compte) => string 'bob'
  protected 'solde' => float 185

C:\wamp64\www\LaPooPHP\3 Héritages\index.php:20:
object(CompteEpargne)[2]
  private 'tauxInterets' => int 1
  private 'titulaire' (Compte) => string 'bob'
  protected 'solde' => float 186.85
```